基于整数线性规划的 VLIW DSP 指令分簇调度 *

周　鹏[1,2]，刘纯纲[1,2]，郑启龙[1,2]

(1. 中国科学技术大学，计算机科学与技术学院，合肥 230026; 2. 高性能计算安徽省重点实验室，合肥 230026)

摘　要： 在分簇 VLIW DSP 上，指令分簇是一项对程序性能有重要影响的编译优化，但现有的指令分簇算法只能处理顺序的程序区域，且难以获得最佳的分簇方案。针对这些问题，提出一种基于整数线性规划的统一指令分簇与指令调度的方法。该方法使用零一决策变量表示函数中指令的分簇、指令的局部调度、以及簇间传输指令的全局调度，并将指令之间的依赖关系和对处理器资源的竞争关系构造为线性约束，最终得到一个以最小化函数的估计执行时间为目标的整数线性规划模型。实验结果表明，求解该模型得到的分簇调度方案对程序性能的优化显著强于现有算法，并且求解模型所耗费的时间是可接受的。

关键词： DSP；VLIW；指令分簇；指令调度；整数线性规划

中图分类号： TP314　　doi: 10.19734/j.issn.1001-3695.2022.03.0120

Integer linear programming based instruction cluster assignment and schedule of VLIW DSP

Zhou Peng[1,2], Liu Chungang[1,2], Zheng Qilong[1,2]

(1. *University of Science & Technology of China*, *College of Computer Science & Technology*, *Hefei* 230026, *China*; 2. *Key Laboratory of High Performance Computing*, *Anhui Province*, *Hefei* 230026, *China*)

Abstract: Cluster assignment is a compiler optimization that plays an important role in performance of programs. However, the existing algorithms only work on straight-line program area, and is difficult to be optimal. Aiming at these problems, this paper proposed a unified cluster assignment and instruction scheduling method based on integer linear programming. This method used zero-one decision variables to represent cluster assignment, local instruction scheduling and global scheduling of inter-cluster transfer instructions, and formulated dependency and processor resources competition between instructions into linear constraints, eventually got an integer linear programming model whose objective is to minimize the estimated execution time of the function. Experimental results show that the cluster assignment and scheduling scheme from solving the model significantly outperforms the existing algorithms on accelerating programs, and the time required to solve the model is acceptable.

Key words: DSP; VLIW; cluster assignment; instruction scheduling; integer linear programming

0　引言

许多数字信号处理器(digital signal processor, DSP)采用超长指令字(very long instruction word, VLIW)架构来提高指令级并行(instruction level parallelism, ILP)。VLIW 架构 DSP 一般发射宽度较大，需要大量的运算设备和寄存器文件。让所有运算设备与所有寄存器直接连通会使处理器的设计复杂度较高，且功耗较大[1]。解决这个问题的一个方法是将运算设备和寄存器分为多个簇，每簇中运算设备与寄存器数目相同，簇内的运算设备能直接访问内部寄存器文件，不同簇之间通过总线连通，跨簇访问数据需要将其他簇的数据复制到本地簇内。这样的分簇架构中，簇内的数据访问延迟低，功耗低，而不同簇之间的数据访问延迟高，功耗高。

在分簇架构的处理器上，需要确定指令使用哪一个簇的运算资源，这一操作称为分簇(cluster assignment)。在一些早期的处理器上，分簇由硬件自动完成，例如文献[2]，在现代的分簇 VLIW DSP 上，分簇由编译器完成，例如由中国电子科技第 38 研究所研发的 HXDSP。分簇的目的是让程序充分利用不同簇上的运算资源，同时避免过多的簇间通信，从而提高程序的指令级并行度，这使得分簇与指令调度这项编译优化紧密相关。在早期的工作中，分簇是一项独立的优化[3~6]，分簇完成后再进行指令调度，更近的一些工作则同时进行指令的分簇与调度[7~10]。

当前的各种分簇方法在基本块或者 trace[11]这样的无分支的程序区域上进行分簇，不同的方法之间的主要区别在于采用了不同的启发因子确定指令被分派的簇。这些方法欠缺对全局数据流的处理，忽视了不同基本块中的指令之间的簇间通信开销，尽管在 trace 中分簇考虑了多个基本块，但在很多程序中，不同分支的跳转情况难以估计，不容易构造 trace。另一问题是基于启发式决策的方法并不能保证得到最优的分簇方案，启发式方法的优点在于算法执行速度快，但在 DSP 的应用领域中，耗费更多的编译时间，实现更好的分簇方案，获得更高的执行性能往往是可以接受的。

组合优化是一类寻找问题最优解的方法，其中整数线性规划是一种形式简单，但被广泛使用的方法，在编译优化领域中也有不少应用，例如指令调度[12, 13]、超字并行[14]。考虑到当前的分簇算法的不能保证最优的现状，使用整数线性规划进行分簇，进一步提高指令分簇的质量是比较好的选择。

本文的主要贡献为：提出了一种 VLIW DSP 上的指令分簇与调度的整数线性规划建模方法，将整个函数中每条指令的分簇，每个基本块的指令调度，以及分簇导致的簇间传输指令的插入位置和调度，表示为一个整数线性规划模型。相比于现有的分簇方法，本文的方法以函数为单位进行分簇，解决了全局数据流之间的簇间传输指令的插入问题，并且求

收稿日期： 2022-03-03；**修回日期：** 2022-05-17　**基金项目：** 国家核高基重大专项资助项目(2012ZX01034-001-001)

作者简介： 周鹏(1994-)，男，四川广安人，硕士研究生，主要研究方向为编译优化(pzzp@mail.ustc.edu.cn)；刘纯纲(1994-)，男，安徽安庆人，硕士研究生，主要研究方向为编译优化；郑启龙(1969-)，安徽合肥人，副教授，硕士，主要研究方向为深度学习平台与应用研究.

chinaXiv:202206.00065v1

ChinaXiv合作期刊

解得到的分簇调度方案是(在估计的代价模型下)最优的，此方法可以直接实现在其他程序区域上，例如只考虑循环内部的代码，基本块，或者 trace 等，当实现在无分支的程序区域上时，执行时间可以准确地评估，得到的分簇调度方案便是真正上最优的。

最早的指令分簇方法由 Ellis 等人在 Bulldog 编译器[3]中实现，称为 BUG(bottom up greedy)算法，此方法基于一项称为完成时刻(completion cycle)的启发因子进行分簇，指令关于某个簇的完成时刻定义为：将指令分派到该簇执行完成后，并将数据传输到流依赖后继所在的候选簇的最晚时刻。BUG 算法从每个无后继的指令节点开始，逆向深度优先遍历数据流图，每次选择使指令完成时刻最小的簇作为当前指令的候选簇，依次对所有依赖前驱分簇，所有前驱分簇完毕后再将当前指令分派到某一个完成时刻最小的簇。

BUG 算法贪心地分簇并不能保证将所有位于关键路径上的指令分派到同一个簇，而且基于最小化完成时间的分簇可能导致首次深度优先遍历数据流图时将一部分有数据流依赖的指令分散到不同簇。Lowney 等人[4]在实现 BUG 算法时进行了一些改进，他们的方法将数据流图分为多个部件，每个部件内部并行度较低，数据共享度高，在 BUG 算法中计算完成时刻时，若两个相关部件(一个的数据被另一个使用)存在于不同的簇中，则在完成时刻上施加一个较大的惩罚值，从而实现若干相关的部件分派到同一个簇，避免簇间通信代价过大。

Desoli 等人[5]采用了一个多阶段的迭代算法实现分簇，首先采用类似 BUG 的算法得到初步的分簇，但分簇的依据是最小化簇间通信量，然后迭代地进行列表调度[15]，根据执行时钟周期数，寄存器压力，簇间通信量调整分簇，直到稳定。列表调度是最为广泛使用的指令调度框架，它根据一些启发因子，例如指令在数据依赖图中的高度，后继的数目等，对数据依赖图进行带优先级的拓扑排序，按照此拓扑序依次尽可能早的发射每一条指令。

Lapinskii 等人[6]的方法按照最迟调度时刻，松弛度(最迟调度时刻与最早调度时刻之差)，数据流后继数目这三个因素对指令排序，按照这个顺序访问每条指令，将指令分派到使计算设备使用代价，总线使用代价，簇间通信代价的带权和最小的簇。在完成初步分簇后，此方法迭代地对不同簇的边缘指令(与其他簇存在通信的指令)进行“扰动”，尝试将其分派到其他簇能否得到更优的结果，进一步提高分簇效果。

Ozer 等人[7]首次提出了统一指令分簇与调度的方法，称为 UAS(unified assignment and scheduling)。UAS 在列表调度的框架下进行，根据指令的数据流依赖前驱指令的完成时间(UAS 称为 completion-weighted predecessor, CWP)，在 BUG 称为开始时刻(Start Cycle)，当前指令开始时刻等于依赖前驱完成时刻的最大值)决定当前待调度指令的被分派的簇，并尽可能早的发射当前指令和其所需的簇间传输指令。

Kailas 等人[8]提出的 CARS 代码生成框架基于列表调度实现了统一了指令分簇和调度，以及寄存器分配的算法，在 CARS 中，指令分簇依然采用了开始时刻作为分簇的启发因子。

Zhang 等人[9]提出了一个精心构造的启发因子，称为 l_{max}-succesor-tree-consistent deadline，定义为对指令及其所有后继的任意一种可行的分簇调度方案(每条指令的发射时刻都不超过给定的上界)中该指令的最晚的执行完成时间。这个方法在列表调度框架下，选择此启发因子最小的就绪指令，将其分派到能尽早发射的簇上，并确定发射时刻。

Porpodas 等人[10]提出的 LUCAS 算法综合了开始时刻与完成时刻作为启发因子，在列表调度框架下，当面临拥塞(就绪指令数目>发射宽度×簇间通信延迟)或高松弛度(松弛度>发射宽度×2(簇间通信延迟-1)，松弛度为指令最迟调度时刻减去最早调度时刻)时选择使开始时刻最小的分簇，否则选择使指令完成时刻最小的分簇。

王玉林等人[16]则使用模拟退火算法进行分簇与调度，将调度后的时钟周期作为反馈，迭代的提高分簇质量。一些工作将分簇与循环的模调度一同实现，在对循环软件流水化时考虑指令的分簇，例如文献[17, 18]，这些方法能有效的提升程序的性能，但循环软件流水要求循环内无分支，或者分支情况比较简单，能使用 If-Conversion[19]转换为谓词指令再实现软件流水[20]，限制较大。

1　总览

本文的方法实现在编译器令选择之后，寄存器分配之前，以整个函数为输入，函数表示为控制流图，每个基本块表示为数据依赖图。

整数线性规划指变量取值为整数的线性规划，定义为，给定一系列整数变量 $\boldsymbol{x}=(x_1,x_2,\ldots,x_n)^T$，称为决策变量，在满足一系列不等式 $\boldsymbol{Ax}\ge b$，以及 $x_1,x_2,\ldots,x_n\ge 0$ 的条件下，使线性函数 $c^T\boldsymbol{x}$ 最小，其中 A 是一个实数矩阵，c 是一个实数向量。决策变量仅取值为 0 或 1 的整数线性规划称为零一规划，是一种常见的建模方式。

本文中指令分簇与调度都使用零一规划来建模，主要决策变量以及相关符号见表 1，约束规则可以粗略地分为三类，包括指令出现次数约束、资源约束和依赖约束，优化目标为每个基本块执行时间的带权和，权值为基本块的估计执行次数，基本块执行时间使用最后一条指令的发射时刻表示。

表 1　文中主要符合及其表示意义

Tab. 1　Symbols in this paper and its meaning

符号	意义
$\rho_{i,m}$	零一变量，指示指令 i 是否被分派到编号为 m 的簇。i 为函数中任意需要分簇的指令(如跳转，NOP 等指令不需要分簇)，$1\le m\le M$ 。
$\tau_{B,i,t}$	零一变量，指示基本块 B 中的指令 i 是在时刻 t 发射。B 为函数中的任意基本块，i 为 B 中任意指令，$1\le t\le T(B)$ 。
$\eta_{B,t}$	零一变量，指示基本块 B 中时刻 t 是否被占用，B 为函数中的任意基本块，$1\le t\le T(B)$ 。
$\delta_{i,j}$	零一变量，指示指令 i,j 是否被分派到同一个簇。i,j 为任意两个存在反依赖或输出依赖的指令。
M	目标平台中簇的数目。
$T(B)$	基本块 B 执行的时钟周期数的一个估计上界。
N	VLIW 长度。
$W(B)$	基本块 B 的执行次数期望。
$I(B)$	基本块 B 中的非簇间传输指令集合
$X(B)$	基本块 B 中所有可能的簇间传输指令集合
$RES(i,k,r)$	指令 i 在流水线第 k 阶段占用机器资源 r 的数目
$M(r)$	目标机器中资源 r 的数目
$x(v,m_1,m_2)$	将变量 v 从簇 m_1 传输到簇 m_2 的簇间传输指令。
$cluster(i)$	整数规划中关于决策变量的表达式，值为指令 i 所在的簇编号。
$cycle(B,i)$	整数规划中关于决策变量的表达式，值为基本块 B 中指令 i 的发射时刻。
$occur(B,x)$	整数规划中关于决策变量的表达式，取值 0 或 1，指示簇间传输指令 x 是否在出现基本块 B 中。

2　簇间传输指令

考虑任意变量 v，它可能因为定义它的指令被分派到不同簇而出现在不同簇，使用 v 的指令也可能被分派到其他不

chinaXiv:202206.00065v1

ChinaXiv合作期刊

同的簇，因此需要簇间传输指令来传递 v。传输变量 v 所有可能的簇间传输指令 $x(v,m_1,m_2)$ 共有 $M(M-1)$ 种。这些簇间传输指令可能被调度到定义与使用 v 之间的任意一条路径上的任意基本块中，为了表示这些调度结果，对于每种簇间传输指令 $x(v,m_1,m_2)$，在这些路径中的每个基本 B 中，决策变量 $\tau_{B,x(v,m_1,m_2),t}$ 都有定义。

例如，对于图 1 的控制流图，定义使用变量 v 的路径包括，A→E, A→E→F, B→C→E, B→C→E→F, B→D→F 这 5 条路径，因此在 A、B、C、D、E、F 上关于传递 v 的簇间传输指令的决策变量 τ 都应当有定义，否则无法表达所有可能的簇间传输指令插入方案，图 2~4 显示了几种分簇与簇间传输指令插入方案(图中簇间传输指令形如 Xv = Yv，表示将 v 从簇 Y 传递到 X)，可见簇间传输指令可以出现在图中每个基本块中。

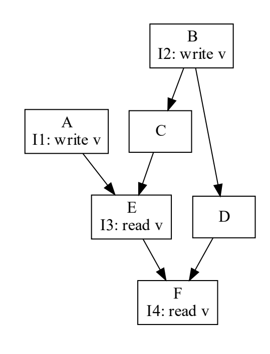


图 1　控制流图

Fig. 1　Control flow graph

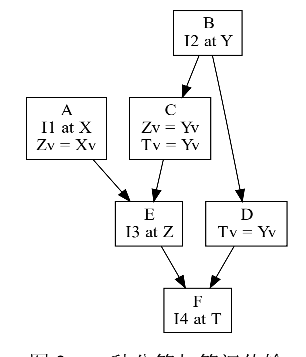


图 2　一种分簇与簇间传输指令插入方案

Fig. 2　A cluster assignment and inter-clustertransfer instructions insertion scheme

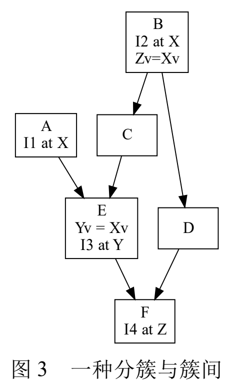


图 3　一种分簇与簇间传输指令插入方案

Fig. 3　A cluster assignment and inter-cluster transfer instructions insertion scheme

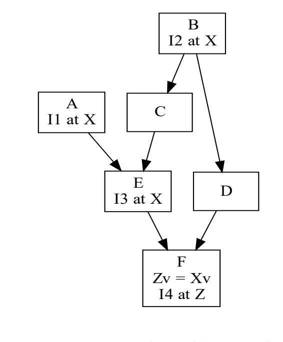


图 4　一种分簇与簇间传输指令插入方案

Fig. 4　A cluster assignment and inter-cluster transfer instructions insertion scheme

但簇间传输指令只出现在输入函数中已有的基本块中并不一定是最优方案。考虑如图 5 所示的控制流图，当 I1 和 I4 被分派到不同簇时，若将簇间传输指令插入在 A 中则会导致在程序执行路径为 A→C 的情况下会执行一条多余的指令，如图 6 所示，而若插入在 D 中，则当程序执行路径为 B→D 时，则执行该簇间传输会导致数据错误，如图 7 所示。这种情况的解决方案是将簇间传输指令插入在路径 A→D 之间新构建的基本块中，如图 8 所示，确保仅在程序执行该路径时簇间传输指令才会被执行。尽管这样可能新的引入跳转指令，导致较长延迟，但若存较多的簇间传输指令插入，重复执行无用指令浪费的时钟周期可能更大。

重复执行与额外的跳转开销之间的权衡会在整数线性规划求解中得到最佳的选择，但必须保证解空间中存在这样的方案。因此，对于任意控制流图的边，若其起点有多条出边，终点有多条入边，则需要在该边上构建一个新的基本块，若该边是直接下落边(即前后两个基本块地址连续)，那么新插入的基本块为空基本块，否则其中应当存在一条跳转指令，跳转目标是该边的后继，这条跳转指令是否出现在最终的目标代码中依赖于新的基本块中是否有簇间传输指令；而若起点只有一条出边或终点只有一条入边，那么簇间传输指令可以插入在起点块或终点块中，插入在新构建的块上只会带来更大的开销。

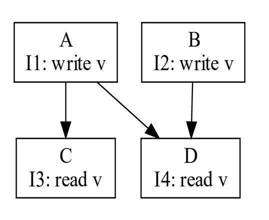


图 5　控制流图

Fig. 5　Control flow graph

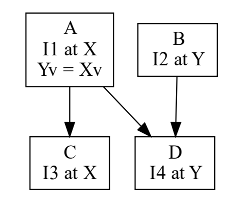


图 6　执行多余指令的情况

Fig. 6　Case of execution redundant instruction

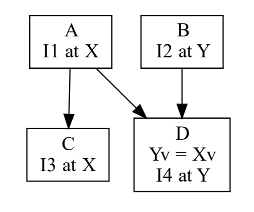


图 7　引起数据错误的情况

Fig. 7　Case of data error

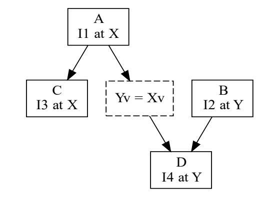


图 8　簇间传输指令插入在新的基本块中的情况

Fig. 8　Case of inter-cluster transfer instruction inserting in new basic block

3　约束

3.1　指令出现次数约束

最基本的，待分簇的指令必须且只能被分派到一个簇中，对于任意待分簇指令 i，有如下约束规则：

$$\sum_{m=1}^{M}\rho_{i,m}=1 \tag{1}$$

在此约束的保障下，根据零一变量的特性，指令 i 被分派到的簇编号可以记为

$$cluster(i):=\sum_{m=1}^{M}m\times\rho_{i,m} \tag{2}$$

同理，函数中原有的基本块 B 中的任意 i 必须且只能被调度到某一个时钟周期：

$$\sum_{t=1}^{T(B)}\tau_{B,i,t}=1 \tag{3}$$

基本块 B 中指令 i 被调度的时钟周期可以记为

$$cycle(B,i):=\sum_{t=1}^{T(B)}t\times\tau_{B,i,t} \tag{4}$$

簇间传输指令被调度的情况依赖于对应变量的定义与使用的指令的分簇情况，可以总结为以下四条规则：

对于任意变量 v，当任意两条读写变量 v 的指令 i,j 被分派到任意不同的两个簇 m_1,m_2 时，簇间传输指令 $x(v,m_1,m_2)$ 必须在它们之间的每条路径 P 中的某个基本块内被调度。容易验证，此约束可以表示为如下形式：

$$\sum_{B\in P}\sum_{t=1}^{T(B)}\tau_{B,x(v,m_1,m_2),t}\ge\rho_{i,m_1}+\rho_{i,m_2}-1 \tag{5}$$

这条约束使簇间传输指令总是从定义变量的指令所在的簇读取数据，排除了簇间传输指令“接力”传递的情况。尽管“接力”方案也是合法方案，但在大多分簇 VLIW DSP 上，“接力”方案不可能是唯一的最优方案。从数据流依赖情况

chinaXiv:202206.00065v1

ChinaXiv合作期刊

来看，簇间传输指令从其他簇间传输指令的结果中读取数据的最早发射时刻总是晚于直接从定义变量的指令所在的簇读取数据。从资源占用方面来看，只有簇间传输指令占用簇间传输总线这一机器资源，且簇间传输指令只占用该资源，因此，无论簇间传输指令从哪个簇读取资源，其资源占用情况都对非簇间指令不构成竞争关系，不会在资源方面影响调度，另一方面，由于总线的特性，只有从同一个簇读取同一数据的簇间传输指令可能在同一个周期发射，因此所有簇间传输指令全都从定义变量的指令所在的簇读取数据能得到的结果总是优于或等于多个簇间传输指令接力的情况，因此，此约束并不会限制降低分簇质量。

对于任意变量 v，设定义它的指令为 i，在控制流图中，任意一条从 i 到使用 v 的指令的路径 P 上，对于任意簇 m，仅当指令 i 没有被分派到簇 m 时，将数据传输到 m 的簇间传输指令才允许出现，且最多在某个基本块中某个时刻出现一次。此约束为

$$\rho_{i,m}+\sum_{B\in P}\sum_{t=1}^{T(B)}\sum_{1\le m_1\le M}\tau_{B,x(v,m_1,m),t}\le 1 \tag{6}$$

这一项约束既限制了同一个变量的相同簇间传输指令最多出现一次，还排除了同一条执行路径上存在多个向某个簇中的变量写入数据的不合法方案，如图 7 的情况。

在定义、使用变量 v 的任意一条路径上的每个基本块 B 中，只有当使用 v 的指令 j 被分派到簇 m 时，才允许簇间向 m 传输数据的传输指令 $x(v,_,m)$ 在该基本块中出现：

$$\sum_{t=1}^{T(B)}\tau_{B,x(v,_,m),t}\le\rho_{j,m} \tag{7}$$

同理，在定义、使用变量 v 的任意一条路径上的每个基本块 B 中，只有当定义 v 的指令 i 被分派到簇 m 时，才允许簇间从 m 传输数据的传输指令 $x(v,m,_)$ 在该基本块中出现：

$$\sum_{t=1}^{T(B)}\tau_{B,x(v,m,_),t}\le\rho_{i,m} \tag{8}$$

簇间传输指令 x 是否出现在基本块 B 可以记为

$$occur(B,x):=\sum_{t=1}^{T(B)}\tau_{B,x,t} \tag{9}$$

在任意一个新插入的基本块 B 中，只可能存在簇间传输指令和一条跳转指令 j，当存在其他簇间传输指令被调度，跳转指令 j 必须在 B 中被调度，此约束可以表示为如下形式：

$$T(B)\sum_{t=1}^{T(B)}\tau_{B,j,t}\ge\sum_{x\in X(B)}occur(B,x) \tag{10}$$

反之，若 B 中不存在簇间传输指令，则 B 可以整个删去，这一条规则不需要表达成约束，因为多一条跳转指令不可能是最优解，会在求解过程中被排除掉。

同一周期发射的指令数目不能超过 VLIW 长度，对于基本块 B 中的调度，在任意时刻 t，有：

$$\sum_{t=1}^{T(B)}\tau_{B,j,t}\le N\eta_{B,t} \tag{11}$$

上式除了约束同一时钟周期发射的指令条数，也确保若时刻 t 存在指令发射，变量 $\eta_{B,t}$ 为 1。但这并不够，为了正确统计基本块 B 中指令调度结果中最后一条指令的发射时刻，必须确保流水线停顿的周期 t 对应的变量 $\eta_{B,t}$ 也为 1，还需要增加如下约束：

$$\forall 2\le t\le T(B):\eta_{B,t-1}\ge\eta_{B,t} \tag{12}$$

此约束也使得指令总是被调度到最早的连续若干个时钟周期。

3.2　依赖约束

基本块中指令之间的依赖关系包括三种数据依赖，分别是流依赖，反依赖输出依赖，另一项依赖关系是，基本块中其他所有指令不能在跳转指令(若存在)之后执行，简单起见，这一项依赖在本文中简称为控制依赖，值得一提的是，广泛使用的术语“控制依赖”有另外的意义[21]。

依赖约束是指，存在数据依赖或控制依赖的两条指令在被调度后，两者所发射的时刻之差应该大于等于依赖延迟。

基本块 B 中，指令 i,j 之间的依赖若与簇内寄存器无关，或是控制依赖时，依赖约束可以统一表示为

$$cycle(B,i)-cycle(B,j)\ge L \tag{13}$$

其中 L 表示依赖延迟，在 VLIW DSP 中，一般控制依赖、反依赖为延迟为 0，输出依赖为 1，流依赖与流水线长度有关。

当反依赖或输出依赖由簇内寄存器传递时，若依赖的前驱或后继被分派到不同的簇时，则依赖自然地被消除，依赖的前后两条指令可以任意调度，否则就需要满足延迟约束，两种情况统一的约束规则可以用大 M 法如下表示：

$$cycle(j)-cycle(i)+2T(B)(1-\delta_{i,j})\ge L \tag{14}$$

$\delta_{i,j}$ 表示指令 i,j 是否被分派到同一个簇，它的语义还需要添加如下约束来得到保证：

$$\begin{aligned}&a_{i,j}+\delta_{i,j}+b_{i,j}=1\\&cluster(i)-cluster(j)\ge -Ma_{i,j}+b_{i,j}\\&cluster(i)-cluster(j)\le Mb_{i,j}-a_{i,j}\end{aligned} \tag{15}$$

上式中还引入了两个辅助的零一变量 $a_{i,j},b_{i,j}$，分别表示 i,j 之间簇的编号关系是小于或大于。

流依赖涉及簇间传输指令，考虑指令 i,j 分别定义、使用变量 v，对于任意簇 m_1,m_2，仅当 $x(v,m_1,m_2)$ 被调度到 i 所在的基本块 B 的某个时刻，且 i 被分派到簇 m_1，才有 $x(v,m_1,m_2)$ 流依赖于 i，即：

$$\begin{aligned}&cycle(B,x(v,m_1,m_2))-cycle(B,i)+\\&2T(B)(2-occur(B,x(v,m_1,m_2))-\rho_{i,m_1})\\&\ge L_{flow}\end{aligned} \tag{16}$$

同理，仅当 $x(v,m_1,m_2)$ 被调度到 j 所在的基本块，且 j 被分派到簇 m_2，j 才流依赖于 $x(v,m_1,m_2)$，有：

$$\begin{aligned}&cycle(B,j)-cycle(B,x(v,m_1,m_2))+\\&2T(B)(2-occur(B,x(v,m_1,m_2))-\rho_{i,m_2})\\&\ge L_{flow}\end{aligned} \tag{17}$$

当 i,j 在同一个基本块时，若它们被分派到同一个簇，它们存在流依赖，因此需要添加同式 13 一样的约束，只是将不等式中的延迟换成流依赖的延迟，若它们被分派到不同的簇,则它们之间必然存在簇间传输指令，约束 16，17 确保了它们的延迟关系能得到满足。

3.3　资源约束

合法的指令调度还必须满足：对于机器中的任意资源，譬如 ALU，立即数通道等，在任意时刻执行的指令使用的每一类资源不能超过机器限制。指令 i 在流水线第 k 阶段使用的资源 r 的数目记为 $RES(i,k,r)$，大多数情况下资源冲突只会发生在少数流水线阶段，不同的指令可能会在占用资源的阶段也可能不同，$RES(i,k,r)$ 只在这些会冲突的阶段上有定义。

分簇 DSP 的资源可以被划分为两类。一类是全局资源，由整个 DSP 共用的，指令使用该资源的情况与指令所在的簇无关；另一类是簇内资源，只能由簇内的指令使用，且每个簇拥有相同数目。值得注意的是，簇间数据总线也是簇内资源，表示为每个簇中的若干个读写端口。

考虑基本块 B 的调度，对于全局资源 r，在任意时刻 t 执行的所有指令使用资源 r 的总和不超过机器限制，有：

$$\sum_{i\in B}\sum_{k\le t}\tau_{B,i,t-k}\times RES(i,k,r)\le M(r) \tag{18}$$

簇内资源中，簇间读写端口需要单独考虑。当 r 是簇间总线的读端口时，对于任意簇 m，仅有从 m 读取数据的簇间传输指令会占用，设可能在 B 中被调度的这些簇间传输指令为 $X(B,m,_)$，簇间传输指令使用总线的约束如下：

chinaXiv:202206.00065v1

ChinaXiv合作期刊

$$\sum_{i\in X(B,m,_)}\sum_{k\le t}\tau_{B,x,t-k}\times RES(x,k,r)\le M(r) \tag{19}$$

类似的，当资源 r 是簇间总线的写端口，对于任意簇 m，设在可能在 B 中被调度的、目标是 m 的簇间传输指令为 $X(B,_,m)$，约束为

$$\sum_{i\in X(B,_,m)}\sum_{k\le t}\tau_{B,x,t-k}\times RES(x,k,r)\le M(r) \tag{20}$$

对于簇内的其他资源，资源使用情况则与待分簇指令自身分派的簇有关，资源约束为

$$\sum_{i\in X(B,_,m)}\sum_{k\le t}(\rho_{i,m}\wedge\tau_{B,x,t-k})\times RES(x,k,r)\le M(r) \tag{21}$$

上式中的 $\wedge$ 表示零一变量的逻辑合取，通过引入额外的一个零一变量，并添加约束 $2(\alpha\wedge\beta)\ge\alpha+\beta,(\alpha\wedge\beta)\le\alpha,(\alpha\wedge\beta)\le\beta$ 得到线性表达。

4　优化目标

分簇调度的目标是使函数执行时间最短，但程序的执行时间是不可判定问题，只能近似。基本块在前文约束的保证下，基本块 B 的最后一条指令的发射时刻为

$$\sum_{t=1}^{T(B)}\eta_{B,t} \tag{22}$$

设输入的函数为 F，优化目标为

$$\sum_{B\in F}W(B)\sum_{t=1}^{T(B)}\eta_{B,t} \tag{23}$$

在没有其他信息的情况下，估计基本块 B 的执行次数 $W(B)$ 基于一个直观的原则：处于循环中的基本块，循环层次越靠内，执行次数越大，远远大于循环外的块，除了循环的回边外，假设跳转指令的分支概率相同。设分簇调度的函数的控制流图是可规约的，本文采取如下方法估计基本块的执行次数：

忽略控制流图中的回边，若在新插入的块在回边上也一同忽略该块，将控制流图的入口块为执行次数设为 $W_0=1$，广度优先遍历控制流图，基本块的出边平分基本块的次数，基本块的执行次数等于入边次数之和。

识别流图中的自然循环，合并具有相同首节点的循环。设基本块(或边)被包含在循中的次数为 k，那么该块(或边)的次数估计为 $100^k W_0$，循环的回边平分回边的起点的权值，回边上插入的基本块的执行次数等于该边的执行次数。

5　其他细节

5.1　处理函数调用指令

出于简化表达的目的，上文中线性整数规划建模忽略了函数调用指令这一特殊情况。函数调用指令实际执行时间依赖于被调用的函数，无法判断，并且当函数返回时，返回值寄存器中的数据已准备好，即使下一条指令要使用函数的返回数据，也不需要再等待流依赖延迟，前文中的依赖约束不能准确的表达这种情况。将函数指令看做基本块边界能绕过这一问题，但会稍微错失一点并行机会，另一个方法是让函数调用指令的发射占用多个时钟周期，等于最大延迟 L，这样当函数调用指令执行完后所有依赖它的指令都已满足延迟，能直接发射，不用改变前文的建模方式。尽管这样做会使优化目标中对最后一条指令发射时刻的计算不精确，但是并不影响最终目的，即对程序执行时间的近似。在这种方法下，函数调用指令发射占用的第 2 到 L 时钟周期，不应该有其他指令占用，设基本块 B 中的函数调用指令为 c，这项约束为

$$\forall 1\le t\le T(B)-L:$$
$$\sum_{u=t}^{t+L-1}\sum_{i\in B\wedge i\ne c}\tau_{B,i,u}\le(1-\tau_{B,c,t})T(B)N \tag{24}$$

5.2　优化求解速度

由于 DSP 的各簇都是同构的，所以存在许多对称的分簇方案，破除这些对称往往能加快整数线性规划求解，思路为尽量将指令分派到簇编号较小的簇。可以通过添加如下约束不等式来实现目标：

$$\forall 2\le m\le M:\sum_{i\in F}\rho_{i,m-1}\ge\sum_{i\in F}\rho_{i,m} \tag{25}$$

基本块 B 中每条指令实际能发射的时刻范围一般远远小于 1 到 $T(B)$ 约束每条指令的发射周期范围能够大幅降低整数线性规划求解时的搜索空间。将程序依赖图看做一个 AOE 网，容易求出每条指令 i 的最早发射时间与最晚发射时间。如果不考虑机器资源，显然指令只能在最早发射时间与最晚发射时间之间被发射，但由于分簇可能插入簇间传输指令，以及 DSP 的发射数限制，实际上指令合法的发射时刻可能会大于该最晚发射时间，因此实际上使用如下方法保守的估计指令发射时刻的区间：

a) 对于任意基本块，遍历不含簇间传输指令的程序依赖图，计算最早发射时刻：

$$et(i)=\begin{cases}0 & if\ \ pred(i)=\varnothing\\ \max\limits_{j\in pred(i)}(et(j)+lat_{i,j}) & otherwise\end{cases}$$

其中 $pred(i)$ 分别表示指令 i 的在图中的所有直接间接前驱，$lat_{i,j}$ 表示指令 i,j 之间的延迟。

b) 遍历包含簇间传输指令的依赖图，使用步骤 a)中的公式，以及结果，计算每条簇间传输指令的最早发射时刻。

c) 计算指令发射时刻的下界：

$$lb(i)=\max\left(et(i),\left\lfloor\frac{|pred(i)|}{N}\right\rfloor\right)$$

d) 遍历不含簇间传输指令的程序依赖图，设当前处理的基本块为 B，计算指令的最晚发射时刻：

$$lt(i)=\begin{cases}T(B) & if\ \ succ(i)=\varnothing\\ \min\limits_{j\in succ(i)}(lt(j)-lat_{i,j}) & otherwise\end{cases}$$

其中 $succ(i)$ 表示指令 i 的所有直接间接后继。此处的最晚发射时刻 lt 与 AOE 网络中的定义并不一样，实际上是反方向计算的最早发射时刻。

e) 遍历包含簇间传输指令的依赖图，使用步骤 d)中的公式，以及结果计算每条簇间传输指令的最晚发射时间。

f) 计算指令发射时刻的上界：

$$ub(i)=\min\left(lt(i),T(B)-\left\lfloor\frac{|succ(i)|}{N}\right\rfloor\right)$$

对于任意基本块 B，限制非簇间传输指令发射在估计的区间内的约束为

$$lb(i)\le cycle(B,i)\le ub(i) \tag{26}$$

簇间传输指令 x 的发射时刻上界约束与其他指令的相同，但由于簇间传输指令 x 不一定被调度，$cycle(B,x)$ 可能为 0，所以，其发射时刻上界约束为

$$lb(i)\le cycle(B,x)+T(B)(1-occur(B,x)) \tag{27}$$

6　实验评估

本文选用 HXDSP1042 作为算法的验证平台，HXDSP1042 一款典型的分簇 VLIW DSP，由中国电子科技集团第 38 研究所研发，支持 16 发射，有 4 个簇，13 级流水线，各簇之间通过总线通信,已经被大量运用在雷达信号处理上。整数线性规划求解是另一个复杂的话题，超出本文讨论的范围，本文使用求解器 gurobi[22]来实现本文的算法。测试程序选择的是 DSPStone[23]，编译时开启 O2 优化，仅替换指令分簇与指令调度优化，并实现了几种统一分簇与调度的

chinaXiv:202206.00065v1

ChinaXiv合作期刊

方法作为参考，这些算法对测试程序中每个基本块分别进行，而本文的方法实现在整个函数上，测试程序在 HXDSP1042 的仿真器上执行，结果显示了本文的方法相比效果最好的近似算法 LUCAS 在不同的测试程序上有 20%到 35%的性能提升。图 9 显示了几种方法在 DSPStone 中几项测试程序中的归一化执行时间，使用 UAS 算法的结果作为基准 1。其中 UAS-CC 表示使用完成时间作为为启发因子的 UAS 算法，LSTC 表示文献[9]提出的算法，ILP 表示本文的方法。

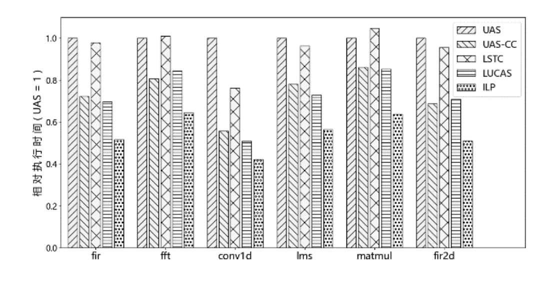


图 9　性能测试结果

Fig. 9　Benchmark results

尽管整数线性规划是 NP 问题，但对于一般程序中的函数，使用分支定界法往往能较快的求解出最优解。本文从在 GSL[24]、FANN[25]、Zlib、DSPStone 中随机选择了共 4000 个函数，在一台 intel Core i5-8300H 的笔记本电脑上测试了本文的算法，结果显示，其中约 81%的函数能在 0.5s 内得到结果，如表 2 所示。

表 2　算法耗时

Tab. 2　Time used by algorithm

耗时范围	最少指令数	最多指令数	平均指令数	函数个数	占总数比例
≤0.02s	4	7	5.2	361	9.03%
0.02-0.1s	4	34	16.9	648	16.20%
0.1-0.5s	15	139	47.2	2237	55.93%
0.5-2.5s	27	213	92.1	354	8.85%
2.5-10s	20	563	159.5	296	7.40%
>10s	74	1588	410.6	104	2.60%

7　结束语

本文使用整数线性规划方法，实现了分簇 VLIW DSP 上的指令分簇与指令调度，并在 HXDSP1042 进行了实验，结果表明本文方法能有效提高程序的性能，且算法的运行时间是可以接受的，尽管与近似算法还有一定差距，但已经具备了相当的实用性。一个不足之处在于对函数执行时间的估计过于粗糙，这个问题可以通过利用编译器高层的信息，或采用深度方法预测[26]分支概率得到改善。另一个问题是本文的方法的时间开销还有优化空间，其原因在于使用了一个大的整数线性规划模型来表示所有指令的分簇与调度，求解时间开销比较大，但实际上往往存在许多指令的分簇和调度与另外许多指令无关。因此下一步的研究将探索如何将函数分割为多个部分，对每个部分分别建模求解并保证整体结果最优，从而进一步提高求解速度。

参考文献：

[1] Mattson P, Dally W J, Rixner S, *et al.* Communication scheduling [J]. ACM SIGOPS Operating Systems Review, 2000, 34 (5): 82-92.

[2] Kessler R. The alpha 21264 microprocessor [J]. IEEE Micro, 1999, 19 (2): 24-36.

[3] Ellis J R. Bulldog: a compiler for VLSI architectures [M]. Cambridge, MA: Mit Press, 1986.

[4] Lowney P G, Freudenberger S M, Karzes T J, *et al.* The multiflow trace scheduling compiler [M]// Instruction-Level Parallelism. Boston: Springer, 1993: 51-142.

[5] Desoli G. Instruction assignment for clustered VLIW DSP compilers: A new approach [M]. Palo Alto, California: Hewlett Packard Laboratories, 1998.

[6] Lapinskii V S, Jacome M F, De Veciana G A. Cluster assignment for high-performance embedded VLIW processors [J]. ACM Trans on Design Automation of Electronic Systems (TODAES) , 2002, 7 (3): 430-454

[7] Ozer E, Banerjia S, Conte T M. Unified assign and schedule: A new approach to scheduling for clustered register file microarchitectures [C]// Proc of the 31st Annual ACM/IEEE International Symposium on Microarchitecture. Piscataway, NJ: IEEE, 1998: 308-315.

[8] Kailas K, Ebcioglu K, Agrawala A. CARS: A new code generation framework for clustered ILP processors [C]// Proc of the 7th International Symposium on High-Performance Computer Architecture. Piscataway, NJ: IEEE, 2001: 133-143.

[9] Zhang Xuemeng, Wu Hui, Xue Jingling. An efficient heuristic for instruction scheduling on clustered VLIW processors [C]// Proc of the 14th International Conference on Compilers, Architectures and Synthesis for Embedded Systems. 2011: 35-44.

[10] Porpodas V, Cintra M. LUCAS: latency-adaptive unified cluster assignment and instruction scheduling [J]. ACM SIGPLAN Notices, 2013, 48 (5): 45-54.

[11] Fisher J A. Trace scheduling: A technique for global microcode compaction [J]. IEEE Trans on Computers, 1981, 30 (07): 478-490.

[12] Wilken K, Liu J, Heffernan M. Optimal instruction scheduling using integer programming [J]. ACM SIGPLAN Notices, 2000, 35 (5): 121-133.

[13] Kästner D, Winkel S. ILP-based Instruction Scheduling for IA-64 [J]. ACM SIGPLAN Notices, 2001, 36 (8): 145-154.

[14] Mendis C, Amarasinghe S. goSLP: globally optimized superword level parallelism framework [J/OL]. Proc of the ACM on Programming Languages, 2018, 2 (OOPSLA): 1-28. (2018-10-24) [2022-05-07]. https://doi. org/10. 1145/3276480.

[15] Fisher J A. The optimization of horizontal microcode within and beyond basic blocks: An application of processor scheduling with resources [D]. New York: New York University, 1979.

[16] 王玉林，郑启龙. 魂芯分簇 VLIW DSP 上指令调度的优化 [J]. 微型机与应用, 2017 (11): 23-26. (Wang Yulin, Zheng Qilong. Instruction scheduling optimization for clustered VLIW HXDSP [J]. Microcomputer & Its Applications, 2017 (11): 23-26)

[17] Zalamea J, Llosa J, Ayguadé E, *et al.* Modulo scheduling with integrated register spilling for clustered VLIW architectures [C]// Proc of the 34th ACM/IEEE International Symposium on Microarchitecture. MICRO-34. Piscataway, NJ: IEEE, 2001: 160-169.

[18] Aleta A, Codina J M, Sánchez J, *et al.* AGAMOS: A graph-based approach to modulo scheduling for clustered microarchitectures [J]. IEEE Trans on Computers, 2009, 58 (6): 770-783.

[19] MAHLKE S A, LIN D C, CHEN W Y, *et al.* Effective compiler support for predicated execution using the hyperblock [J]. ACM SIGMICRO Newsletter, 1992, 23 (1-2): 45-54.

[20] LLOSA J, GONZÁLEZ A, AYGUADÉ E, *et al.* Swing module scheduling: a lifetime-sensitive approach [C]// Proc of the 22nd Conference on Parallel Architectures and Compilation Technique. Piscataway, NJ: IEEE, 1996: 80-86.

[21] Ferrante J, Ottenstein K J, Warren J D. The program dependence graph and its use in optimization [J]. ACM Trans on Programming Languages

chinaXiv:202206.00065v1

ChinaXiv合作期刊

and Systems (TOPLAS) , 1987, 9 (3): 319-349.

[22] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual [EB/OL]. [2022-05-07]. https://www. gurobi. com.

[23] Institute for Communication Technologies and Embedded Systems. DSPStone [EB/OL]. [2022-05-07]. https://www. ice. rwth-aachen. de/research/tools-projects/closed-projects/dspstone.

[24] Gough B. GNU scientific library reference manual [M]. [S. l.] : Network Theory Ltd. , 2009.

[25] Nissen S. Implementation of a fast artificial neural network library (fann) [EB/OL]. (2003) [2022-05-07]. http://fann. sourceforge. net/report.

[26] Bieber D, Sutton C, Larochelle H, *et al.* Learning to execute programs with instruction pointer attention graph neural networks [C/OL]// Advances in Neural Information Processing Systems. New York, NY: Curran Associates, Inc. , 2020: 8626-8637. https://proceedings. neurips. cc/paper/2020/file/62326dc7c4f7b849d6f013ba46489d6c-Paper. pdf

chinaXiv:202206.00065v1